LE/EECS 2021 4.00   Computer Organization

LAB B REPORT

Michael Williams 211087798

Section E

September 29 2014

The work in this report is my own. I have read and understood York University academic dishonesty policy and I did not violate the senate dishonesty policy in writing this report.

ABSTRACT

In this lab, I was introduced to assembly and assembly coding, and the many ways to translate and compute regular java programming code to MIPS. The purpose was to teach us about the simple manipulative and arithmetic expressions within MIPS, and more importantly, how to take a code in java, and loosely transfer that code for use in another machine, in this case being xspim. We were taught simple basic expressions that allowed us to add, subtract, multiply and divide, print out values to the user, read input from the user and compare values. We were also shown how to make translations of if statements and loops, as well as bit manipulation from lab A, only this time with MIPS. In conclusion, these programs introduced us to the basic expressions of a new coding language, giving us knowledge for our future labs a little more insight into how MIPS and xspim work, separately and together.

EQUIPMENT

-laptop

-java eclipse

-putty

-xming

-xspim

-gedit

METHODS/PROCEDURES

For the most part, the lab was a simple. For programs 1-14, I simply followed the given instructions, and made little alterations to what was provided. Because they were almost entirely just doing exactly as was shown in the lab manual, there wasn’t much problem solving involved. I did have some trouble when it came to programs 15-18. This was due to a few reasons. First, was because bit manipulation involving masks also gave me a little trouble in the previous lab. Second, because I had trouble translating to MIPS and was not entirely sure how to use the given expressions. However, after looking up their uses, it became clear how they worked and when/where to use them. In program 18 specifically, I overcomplicated the program itself, instead of simply using the xor gate to its proper context, as shown in the appendix section.

RESULTS

The results were as expected throughout all the labs. For labs 1-15, this was simply the results described within the program instructions. For labs 16-18, the results were variable, as they were dependent on the output. However, the output did successfully result in accurate responses, despite the issues and problems I faced to compute the programs.

DISCUSSION

There wasn’t much to discuss in terms of this lab, considering this was a lab to introduce us to the basic expressions of the MIPS program. For the most part, everything was straight forward. The only part I had little trouble with was isolating certain bits during the programs that involved bit manipulation with certain expressions, but once I figured out how to implement masks correctly, and make use of the xor and nor gate properly, the issue became easily correctable.

CONCLUSION

In conclusion, the lab was very straight forward, and did not provide much challenge in terms of understanding, or implementation. I achieved everything I sought to learn with this lab, and, with the exception of the last few labs, easily overcame any issues that arised while making the programs, learning the base instructions of MIPS programming and how to translate simple java coding to MIPS , as well as the many processes it involves, and the drastic changes simply changing a single bit can have to given values.

APPENDIX

LabB1

.text

main: #-----------------

li $t0, 60 # t0 = 60

addi $t1, $0, 7 # t1 = 7

add $t2, $t0, $t1 # t2 = t0+t1

#-----------------

addi $v0, $0, 1 #service #1

add $a0, $0, $t2 # printInt

syscall #do print

#------------------

jr $ra #return

LabB2

.globl fini

.text

main: #-----------------

li $t0, 60 # t0 = 60

addi $t1, $0, 7 # t1 = 7

add $t2, $t0, $t1 # t2 = t0+t1

#-----------------

addi $v0, $0, 1 #service #1

add $a0, $0, $t2 # printInt

syscall #do print

#------------------

fini: jr $ra #return

LabB3

.globl fini

.text

main: #-----------------

li $t0, 60 # t0 = 60

addi $t1, $0, 7 # t1 = 7

add $t2, $t0, $t1 # t2 = t0+t1

#-----------------

addi $v0, $0, 11 #service #11

add $a0, $0, $t2 # printInt

syscall #do print

#------------------

fini: jr $ra #return

LabB4 .globl fini

.text

main: #-----------------

li $t0, 60 # t0 = 60

addi $t1, $0, 7 # t1 = 7

add $t2, $t0, $t1 # t2 = t0+t1

sub $t3, $t0, $t1 # t3 = t0-t1

#-----------------

addi $v0, $0, 1 #service #1

add $a0, $0, $t2 # printInt

syscall #do print

#------------------

addi $v0, $0, 11 #service #11

add $a0, $0, ' ' #print space

syscall #do print

#------------------

addi $v0, $0, 1 #service #1

add $a0, $0, $t3 #printInt

syscall #do print

#------------------

fini: jr $ra #return

LabB5

.globl fini

.text

main: #-----------------

addi $v0, $0, 5 #service #5

syscall #readInt

add $t0, $0, $v0 #add Int

addi $t1, $0, 7 # t1 = 7

add $t2, $t0, $t1 # t2 = t0+t1

#-----------------

addi $v0, $0, 1 #service #1

add $a0, $0, $t2 # printInt

syscall #do print

#------------------

fini: jr $ra #return

LabB6

.globl fini

.text

main: #-----------------

addi $v0, $0, 5 #service #5

syscall #readInt

add $t0, $0, $v0 #add Int

addi $t1, $0, 7 # t1 = 7

beq $t0, $t1, XX

sub $t2, $t0, $t1 #t2 = t0-t1

j YY

XX: add $t2, $t0, $t1 # t2 = t0+t1

#-----------------

YY: addi $v0, $0, 1 #service #1

add $a0, $0, $t2 # printInt

syscall #do print

#------------------

fini: jr $ra #return

LabB7

.globl fini

.text

main: #-----------------

addi $v0, $0, 5 #service #5

syscall #readInt

add $t0, $0, $v0 #add Int

addi $t1, $0, 7 # t1 = 7

slt $t3, $t0, $t1 #t3 compares t0 and t1

bne $t3, 1, XX

add $t2, $t0, $t1 #t2 = t0+t1

j YY

XX: sub $t2, $t0, $t1 # t2 = t0-t1

#-----------------

YY: addi $v0, $0, 1 #service #1

add $a0, $0, $t2 # printInt

syscall #do print

#------------------

fini: jr $ra #return

LabB8

.globl fini

.text

main: #-----------------

addi $v0, $0, 1 #service#1

add $a0, $0, $0

loop: slti $t9, $a0, 5

beq $t9, $0, fini

syscall

addi $a0, $a0, 1

j loop

fini: jr $ra #return

LabB9

.globl fini

.text

main: #---------------------------------------

add $v0, $0, 5 #service #5

syscall

add $t0, $0, $0

add $t1, $0, $v0 #t1 = read int

#---------------------------------------

add $t5, $0, $0

loop: slt $t2, $t5, $t1 #check if t5 < t1

bne $t2, 1, done #jump to done if 1

add $t5, $t5, 1 #else, add 1 to t5

add $t0, $t0, $t5 #and add t5 to t0

j loop

#--------------------------------------

done: add $v0, $0, 1 #service #1

add $a0, $0, $t0

syscall #doPrint

fini: jr $ra #return

LabB10

.globl fini

.text

main: #--------------------

addi $t0, $0, 60

addi $t1, $0, 7

div $t0, $t1

mflo $a0

#--------------------

addi $v0, $0, 1 #system#1

syscall #print quotient

addi $v0, $0, 11

add $a0, $0, ' '

syscall

#-------------------------

mfhi $a0

addi $v0, $0, 1

syscall

addi $v0, $0, 11

add $a0, $0, ' '

syscall #print remainder

#---------------------

mult $t0, $t1

mflo $a0 #print least 32 bits

addi $v0, $0, 1

syscall

addi $v0, $0, 11

add $a0, $0, ' '

syscall

#---------------------------------------

mfhi $a0 #print most 32 bits

addi $v0, $0, 1

syscall

addi $v0, $0, 11

add $a0, $0, ' '

syscall

fini: jr $ra #return

LabB11

.globl fini

.text

main: #-----------------

addi $t0, -60 # t0 = 60

sra $a0, $t0, 1

addi $v0, $0, 1

syscall

#-----------------

addi $v0, $0, 11 #service #11

add $a0, $0, ' ' # printInt

syscall #do print

#------------------

sll $a0, $t0, 1

addi $v0, $0, 1

syscall

fini: jr $ra #return

LabB12

.globl fini

.text

main: #-----------------

addi $v0, $0, 5

syscall

add $t0, $0, $v0

#------------------------

sll $t1, $t0, 4

sll $t2, $t0, 1

add $a0, $t1, $t2

#--------------------

addi $v0, $0, 1

syscall

fini: jr $ra #return

LabB13

.globl fini

.text

main: #-----------------

addi $v0, $0, 5

syscall

add $t0, $0, $v0

#------------------------

sll $a0, $t0, 21

srl $a0, $a0, 31

#--------------------

addi $v0, $0, 1

syscall

fini: jr $ra #return

LabB14

.globl fini

.text

main: #-----------------------

addi $v0, $0, 5

syscall

add $t0, $0, $v0

#------------------------

andi $a0, $t0, 1024

#------------------------

addi $v0, $0, 1

syscall

fini: jr $ra #return

LabB15

.globl fini

.text

main: #-----------------------

addi $v0, $0, 5

syscall

add $t0, $0, $v0

#------------------------

add $t5, $0, 0xffff

sll $t5, $t5, 16

ori $t5, $t5, 0xfbff #will create mask of all 1s and 0 in bit #10

and $a0, $t0, $t5

#--------------------

addi $v0, $0, 1

syscall

fini: jr $ra #return

LabB16

.globl fini

.text

main: #-----------------

addi $v0, $0, 5

syscall

add $t0, $0, $v0

#------------------------

addi $t6, $0, 1024

nor $t4, $0, $t6 #will create ~1024

and $a0, $t4, $t0

#--------------------

addi $v0, $0, 1

syscall

fini: jr $ra #return

LabB17

.globl fini

.text

main: #---- -------------------

addi $v0, $0, 5

syscall

add $t0, $0, $v0

#------------------------

add $t5, $0, 0xffff

sll $t5, $t5, 16

ori $t5, $t5, 0xfbff #will create mask of all 1s and 0 in bit #10

and $a0, $t0, $t5

#--------------------

addi $v0, $0, 1

syscall

addi $v0, $0, 11

add $a0, $0, ' '

syscall

#------------------------

lui $t7, 0xffff

ori $t7, $t7, 0xfbff #will create same t5 mask easier

and $a0, $t7, $t0

addi $v0, $0, 1

syscall

fini: jr $ra #return

LabB18

.globl fini

.text

main: #---- -------------------

addi $v0, $0, 5

syscall

add $t0, $0, $v0

#------------------------

# add $t5, $0, 0xffff

# sll $t5, $t5, 16

# ori $t5, $t5, 0xfbff #will create mask of all 1s and 0 in bit #10

# and $a0, $t0, $t5

#------------------------

sll $t1, $t0, 21

srl $t1, $t1, 31 #t1 = bit#10

#--------------------------------------

beq $t1, $0, XX # if t1 = 0, go to XX

lui $t7, 0xffff

ori $t7, $t7, 0xfbff #will create mask with all 1s and a zero

and $a0, $t0, $t7 #replace bit #10 with a 0

j YY

XX: addi $t7, $0, 1024 #will create mask with all zeros and a 1

xor $a0, $t7, $t0 #replace bit 10 with a 1

#------------------------------------------

YY: addi $v0, $0, 1

syscall

fini: jr $ra #return